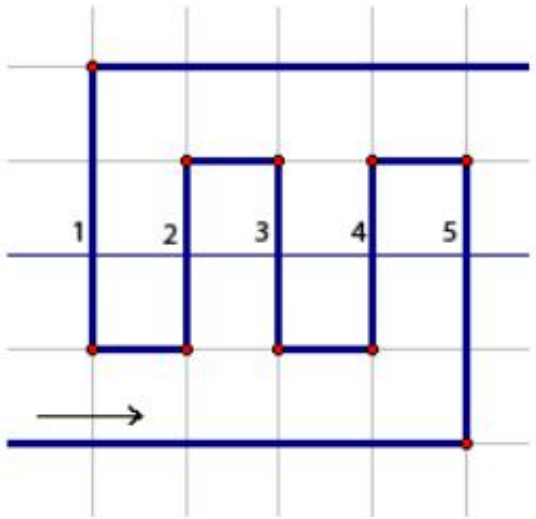
Альтернативный экзамен по дискретной математике. "Меандры"

Выполнил: Варивода С. А.
Преподаватель: Поздняков С. Н.

Постановка задач

- Первая задача: дана перестановка - определить, задаёт ли она меандр.
- Вторая задача: сгенерировать все меандры для n пересечений.

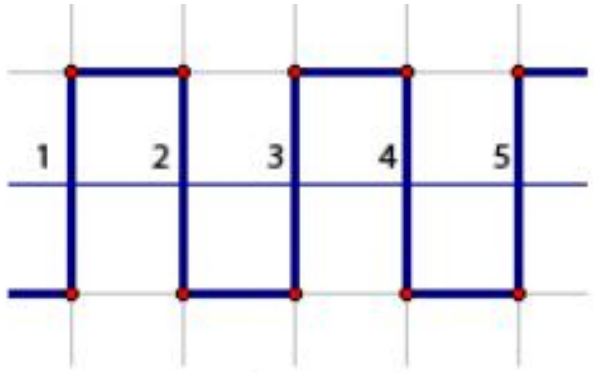


Определение меандра

Задача, которую мы будем анализировать, формулируется так:

Шоссе, идущее с запада на восток, пересекает несколько раз реку, текущую с юго-запада на восток. Занумеруем мосты в порядке их следования вдоль шоссе (с запада на восток). Проплывая под мостами вниз по реке, мы будем встречать их, вообще говоря, в другом порядке. Например, 3,4,5,2,1. Таким образом, эта река определяет перестановку. Другая река могла бы задавать другую перестановку .
Но далеко не любая перестановка чисел (мостов) может быть реализована таким образом. Например, не придумать реку, проходящую через мосты в порядке 2,1,3,4,5.

Будем называть перестановку меандром, если ее можно задать с помощью подходящей реки.



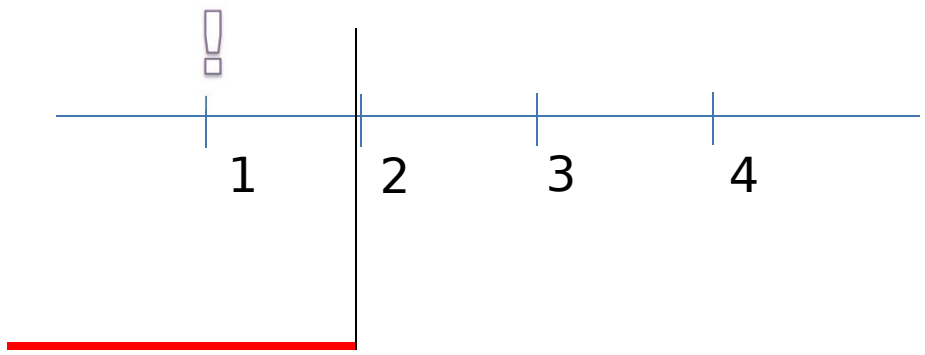
Решение первой задачи

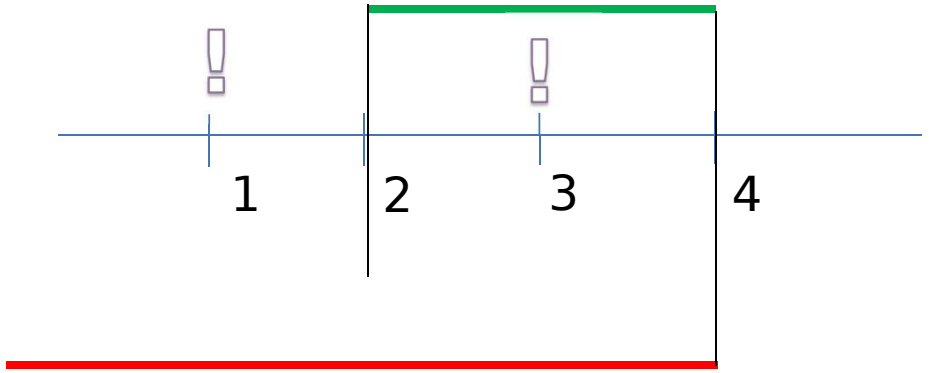
Проведя исследование, я заметил, что меандр должен соответствовать двум условиям:

1) Начинаться с нечетного моста
2) Дальше четные и нечетные мосты должны чередоваться

То есть четность позиции (если считать с 1) должна соответствовать четности номера моста.

Иначе возникают меандры, к которым "не подойти":





Решение первой задачи

Но оговоренных выше условий недостаточно, поэтому я предлагаю следующий алгоритм проверки меандра.

1) Создадим переменную-флаг, которая отмечает "снизу" или "сверху" мы пришли к i-ому элементу.

2) Создадим два массива, верхних и нижних ограниченных отрезков.

3) На каждой итерации записываем в соответствующий массив "заблокированный отрезок"

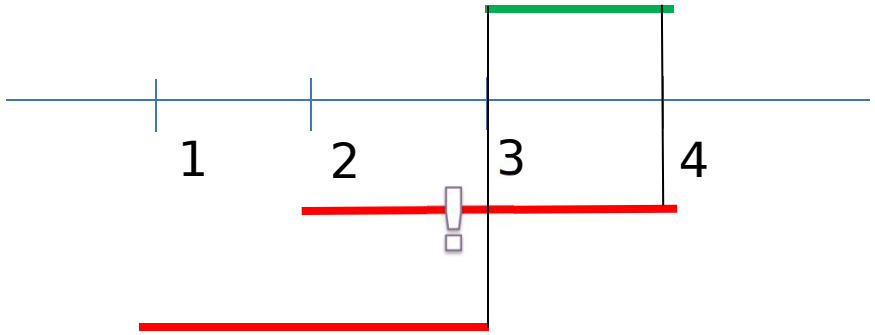
4) Смотрим, попадает ли элемент в "заблокированный отрезок".

5) Если попадает, то проверяем предыдущий элемент, и если он не в отрезке, то это пересечение.

Решение первой задачи

Рассмотрим пример.

Перестановка: 3 4 2 1
Левая граница: 1
Флаг: снизу
Массив нижних отрезков (H): пустой
Массив верхних отрезков (B): пустой



Идем в цикле по перестановке.

1) Т.к. массивы пустые записываем в H (т.к. флаг снизу) [1, 3],
меняем флаг на "вверху" и левую границу на 3.

2) 4 не попадает в отрезки, поэтому просто добавляем в B [3,4],
меняем флаг на "снизу" и левую границу на 4.

3) К 2 мы пришли снизу и попадаем в отрезок содержащийся в H - [1,3].
Значит смотрим на предыдущий элемент. 4 вне этого отрезка.

Это не меандр.

Решение второй задачи (1-ый алгоритм)

Воспользуемся стандартным алгоритмом генерации лексикографических перестановок без повторений.

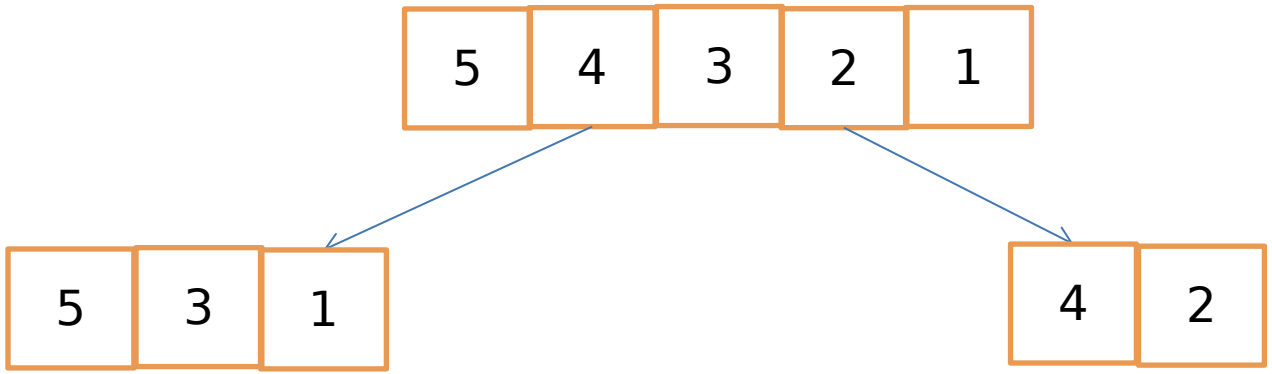
Дана исходная последовательность чисел. Для получения каждой следующей перестановки необходимо выполнить следующие шаги:

1) Необходимо просмотреть текущую перестановку справа налево и при этом следить за тем, чтобы каждый следующий элемент перестановки (элемент с большим номером) был не более чем предыдущий (элемент с меньшим номером).
2) Как только данное соотношение будет нарушено необходимо остановиться и отметить текущее число (позиция 1).
3) Снова просмотреть пройденный путь справа налево пока не дойдем до первого числа, которое больше чем отмеченное на предыдущем шаге.
4) Поменять местами два полученных элемента.
5) Теперь в части массива, которая размещена справа от позиции 1 надо отсортировать все числа в порядке возрастания. Поскольку до этого они все были уже записаны в порядке убывания необходимо эту часть подпоследовательность просто перевернуть.

Сгенерировав новую перестановку, проверяем ее алгоритмом проверки описанным выше.

Решение второй задачи (2-ой алгоритм)

Сложность 1-го алгоритма O(n!).
Чтобы ускорить его я разделил перестановку на 2 массива четных и нечетных чисел.



И в двойном цикле мы делаем перестановки отдельно для каждого массива.
После чего собираем обратно в один массив и проверяем на пересечения.
Данный алгоритм будет генерировать сразу перестановки с чередующейся четностью элементов.

Сравнение алгоритмов

Оба алгоритма генерируют меандры для любого N. Но из-за сложности перебора может быть затрачено большое кол-во времени. Приведу стравнительную таблицу скорости нахождения всех меандров до от 6 до 15 (для меньших нет смысла проверять).

N	Алгоритм 1, миллисекунды	Алгоритм 2, миллисекунды
6	3	3
7	5	6
8	8	10
9	80	19
10	407	30
11	5121	100
12	61310	198
13	>120000	1448
14	>1000000	11948
15	>1000000	110179

Сравнение полученных результатов

Сравнив известные меандры и полученные, мы можем убедиться, что алгоритмы работают верно.
Например, результаты полученные для 6:

1 2 3 4 5 6
1 2 3 6 5 4
1 4 3 2 5 6
1 6 3 4 5 2
1 2 5 4 3 6
1 4 5 6 3 2
1 6 5 2 3 4
1 6 5 4 3 2
3 2 1 4 5 6
3 2 1 6 5 4
3 4 5 2 1 6
5 4 1 2 3 6
5 2 3 4 1 6
5 4 3 2 1 6

Эти результаты совпадают с теоретическими.
Все результаты генераций до 16 можно найти в репозитории.

Используемые технологии

В качестве основного языка программирования выбран Java.

Выбор был сделан опираясь на главное преимущество этого языка - мультиплатформенность.